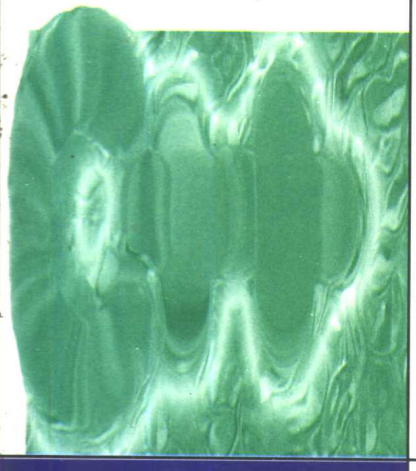
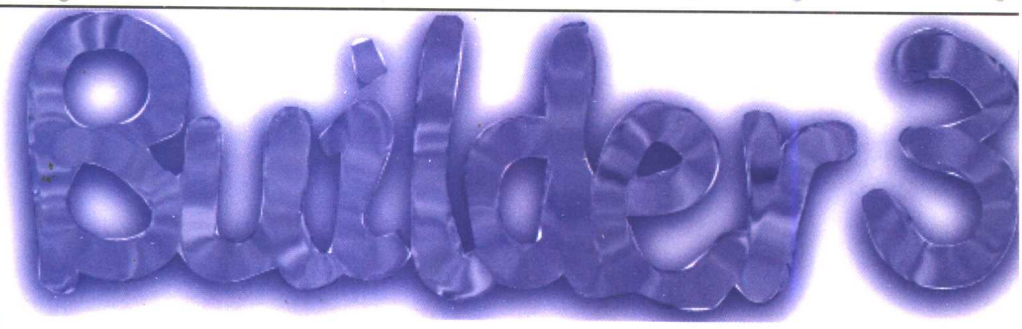
# C++ Builder 3 数据库编程指南

徐新华 著






清华大学出版社
http://www.tup.tsinghua.edu.cn

# C++ Builder 3
# 数据库编程指南

徐新华　著

清华大学出版社

(京)新登字 158 号

## 内 容 简 介

本书讲述的是专题内容,深入介绍了当今最优秀的编程工具 C++ Builder 3 的数据库技术,包括面向对象的编程思想、数据集、数据访问链路、数据控件、QuickReport 报表、TeeChart 图表、决策方、多层 Client/Server、数据库浏览器等内容。

本书内容全面而又不失简洁,既可以作为广大读者学习 C++ Builder 3 的入门指导书,也可以作为程序员编程时的参考手册。



**图书在版编目(CIP)数据**

JS481/10

C++ Builder 3 数据库编程指南/徐新华著.—北京:清华大学出版社,1999
ISBN 7-302-03339-0

Ⅰ.C… Ⅱ.徐… Ⅲ.C 语言-软件工具,C++ Builder 3 Ⅳ.TP311.56

中国版本图书馆 CIP 数据核字(1999)第 03290 号

**出 版 者**:清华大学出版社(北京清华大学校内,邮编 100084)
http://www.tup.tsinghua.edu.cn
**责任编辑**:汤斌浩
**印 刷 者**:北京市清华园胶印厂
**发 行 者**:新华书店总店北京发行所
**开 本**:787×1092 1/16 **印张**:15.75 **字数**:370 千字
**版 次**:1999 年 2 月第 1 版 1999 年 2 月第 1 次印刷
**书 号**:ISBN 7-302-03339-0/TP·1800
**印 数**:0001~5000
**定 价**:24.00 元

# 前言

C++ Builder 3 是 Borland(r)公司 1998 年第一季度推出的拳头产品，是当今世界上最优秀的 C++ 开发工具。C++ Builder 3 支持许多业界最新的技术，最接近 ISO 的 C++ 标准。C++ Builder 3 在测试阶段就已获得了许多权威机构和杂志的极高评价，在市场上正式推出后，好评如潮，购买踊跃。

在中国，C++ Builder 3 同样受到了热烈的欢迎，许多程序员纷纷从其他开发工具转到 C++ Builder 3 上来。他们认为，C++ Builder 3 的开发效率很高，改进了中文处理能力，特别是在面向对象领域，C++ Builder 3 处于领先的地位。

为了帮助广大用户全面、准确地掌握 C++ Builder 3 在数据库方面的编程技术，我们编写了这本《C++ Builder 3 数据库编程指南》，主要是针对那些已初步掌握了 C++ Builder 3 的基本用法，现在需要在数据库方面进一步精通和提高的读者。本书紧紧把握住了 C++ Builder 3 的最基本特征，即面向对象，重点从类这一层次把 C++ Builder 3 的数据库技术讲透。本书的第一章专门讲述了 C++ Builder 3 的面向对象的编程思想和 VCL 结构。本书的内容具有一定的深度，涉及到许多高级编程技巧，并配有大量的程序示例。

本书主要由徐新华执笔，参加编写工作的还有杨健、张羿、杨传才、刘胜、张斌、朱世新、王玮、李东、赵亮、郭平、徐新辉、张莉、顾玉英、顾洪均、刘忠德、徐忠、周杰等。

由于我们的水平有限，再加上时间很紧，尽管我们对书中的内容作了严格的审核和测试，但可能还是难免有一些错误，敬请广大读者不吝赐教。我们谨在此表示感谢。

考虑到 C++ Builder 3 的数据库技术比较复杂，市面上的参考书又很少，为了帮助广大 C++ Builder 3 的用户更好地掌握这门技术，作者愿意为购买此书的读者提供技术咨询。读者可以拨打电话(010)62987260 或者发 E-mail 至 p_inprise@mail.263.net.cn 与作者联系，我们将热情、及时地答复大家提出的问题。

徐新华

1998 年 6 月 15 日

# 本书内容概览

本书讲述的是专题内容，深入介绍了当今最优秀的编程工具 C++ Builder 3 的数据库技术，包括面向对象的编程思想、数据集、数据访问链路、数据控件、QuickReport 报表、TeeChart 图表、决策方、多层 Client/Server、数据库浏览器等内容。

本书共分九章，具体安排如下：

第一章介绍 C++ Builder 3 的面向对象编程思想和 VCL 的结构，这是本书的基础部分。

第二章介绍数据集的公共基类，包括 TDataSet、TBDEDataSet、TDBDataSet、TField 和字段编辑器。

第三章介绍怎样建立数据库访问链路，重点是用 TTable、TQuery、TStoredProc 引入数据集，用 TDatabase 连接数据库，用 TSession 管理 BDE 会话期对象。

第四章介绍怎样用数据控件显示和编辑数据库的数据。几个特别难理解的控件，如 TDBListBox、TDBLookupListBox、TDBCtrlGrid 在这里都讲得非常清晰和透彻。

第五章和第六章分别介绍怎样制作 QuickReport 报表和 TeeChart 图表。

第七章介绍 C++ Builder 3 的决策分析技术。

第八章介绍多层 Client/Server 技术，这是本书的重点，也是 C++ Builder 3 最重要的功能之一。这一章主要分成两个部分：应用服务器和“瘦”客户。

第九章介绍了一个非常有用的实用工具，即数据库浏览器。

本书少数地方仍然采用了 Pascal 的语法，请读者注意与 C++ 的语法对照。

# 目 录

# 第一章 面向对象编程

C++ Builder 3 的最大特点就是面向对象，在面向对象领域，Borland(r)一直处于世界领先的地位。C++ Builder 3 的面向对象主要体现在三个方面：一是编程语言最接近 ISO C++ 标准；二是全面支持基于元件的应用程序开发；三是借助于对象库可实现代码重用。

本章从一些基本概念开始，详细介绍 C++ Builder 3 面向对象的编程思想，并初步介绍 VCL 的结构。C++ Builder 3 中的元件很多，尽管这些元件千差万别，但它们都是从几个公共基类继承下来的，因此，这些元件具有某种程度的相似性。为了节省篇幅，本书按照这样的思路编写，即先介绍公共基类，后面则只介绍每个元件特有的特性和方法。

## 1.1 什么是对象

什么是对象？这先得从什么是类说起。类(Class)是一种数据类型，与一般的数据类型不同的是，类除了包含数据以外，还包含了操纵数据的方法，类把数据和方法封装在一起。在一个类中，可以包含不同类型的数据，这一点与 C++ 的结构相似。

类具有可继承性。如果要创建一个新的类，只要选择一个基类再加上自己的成员，就可以派生出一个新的类。事实上，C++ Builder 3 中所有的类都是从一个共同的基类继承下来的，基类的非私有成员自动成为派生类的成员。类的继承还具有传递性，例如，假设 T3 继承了 T2，而 T2 又继承了 T1，那么可以认为 T3 也继承了 T1。

在 C++ Builder 3 中，所有的类都是从 TObject 继承下来的。TObject 是一个抽象类，它的派生类可以对 TObject 的方法(包括构造和析构)重载。TObject 也被称为默认祖先类。

类只是一种数据类型。要使用类，一般还得声明类的变量(某些特殊的情况下可以直接对类进行操作)。需要指出的是，类的变量和类的实例从严格意义上讲，不完全是一个概念。类的实例(Instance)是一块动态分配的内存区域，一般我们把类的实例称为对象(Object)。同一个类，可以有多个实例存在，每个对象都有自己的数据，但方法是共享的。

类的变量实际上是一个指针，指向类的实例。要访问对象的成员，就要通过类的变量来引用。同一个类可以有多个变量，多个变量可以引用同一个对象。

下面我们还是通过一个典型的程序示例来把类和对象的概念讲清楚。当您使用“File”菜单上的“New Application”命令时，C++ Builder 3 将创建一个新的项目。在默认情况下，这个项目中只有一个空白的 Form 和一个单元文件及头文件。

在 C++ Builder 3 中，Form 就是一个非常典型的对象。Form 的直接基类是 TForm，我们先看看头文件中是怎样声明 Form 的：

```
//---------------------------------------------------------
#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------
class TForm1 : public TForm
{
    __published:
    private:
    public:
    __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------
#endif
```

可以看出,头文件中声明了一个类叫 TForm1,它的基类是 TForm。由于 Form 现在是空白的,也没有建立任何事件句柄,因此,TForm1 中除了构造函数外没有其他数据和方法。

但如果您对类的概念理解得比较透彻的话,您就知道,TForm1 中实际上是有成员的,因为 TForm1 是从 TForm 继承下来的,因此 TForm 的成员也就成为了 TForm1 的成员。

声明了 TForm1 后,头文件中声明了 TForm1 类型的变量 Form1,我们以后就是用 Form1 来引用这个 Form 的。注意,在调用 TForm1 的构造函数之前,也就是说在创建 TForm1 的实例之前,Form1 这个变量基本上是没意义的。

我们再看看单元文件的代码是怎样的:

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
}
    //---------------------------------------------------------
```

由于现在 Form 是空白的,因此单元文件中只有 TForm1 的构造函数。您可以在构造

函数中加入任何代码,但不能引用 Form 本身,因为在构造函数中,类的实例还没有创建好。

假设您向 Form 中加入一个按钮(TButton),名称是 Button1。双击此按钮,C++ Builder 3 就会自动建立一个处理该按钮的 OnClick 事件的句柄。这时候,头文件就变成这样:

```
//---------------------------------------------------------
#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------
class TForm1 : public TForm
{
    __published:
    TButton *Button1;
    void __fastcall Button1Click(TObject *Sender);
    private:
    public:
    __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------
#endif
```

可以看出,头文件在 TForm1 的 __published 部分声明了一个 TButton 类型的变量,名为 Button1,同时声明了一个无返回值的函数 Button1Click 作为处理 OnClick 事件的句柄。

我们再看看单元文件的代码有什么变化:

```
//---------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{

}
//---------------------------------------------------------
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
}
//---------------------------------------------------------
```

可以看出,单元文件唯一的变化就是给出了事件句柄的实体。目前该实体中没有代码,您可以加入任何代码,包括对 Form 的引用。程序示例如下:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Form1->Color = clGreen;
}
```

## 1.2 修改元件的名称

一般来说,您最好在设计期用 Object Inspector 修改元件的名称,而且最好在您刚刚把元件加到 Form 上时就改。在设计期修改元件的名称有一个好处,即 C++ Builder 3 会自动更新源代码中所有引用该元件名的地方,也就是说,您不用担心是否有不一致的地方。例如,您可以把 Form 的名称从 Form1 改为 MainForm,头文件会相应变化:

```
//---------------------------------------------------------
#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------
class TMainForm : public TForm
{
    __published:
    TButton *Button1;
    void __fastcall Button1Click(TObject *Sender);
    private:
    public:
    __fastcall TMainForm(TComponent* Owner);
};
//---------------------------------------------------------
extern PACKAGE TMainForm *MainForm;
//---------------------------------------------------------
#endif
```

可以看出,所有原先是 Form1 的地方现在都改为了 MainForm,包括类名。

完全可以预料,单元文件的代码也改过来了:

```
//---------------------------------------------------------
```

```
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TMainForm *MainForm;
//---------------------------------------------------------
__fastcall TMainForm::TMainForm(TComponent* Owner)
: TForm(Owner)
{

}
//---------------------------------------------------------
void __fastcall TMainForm::Button1Click(TObject *Sender)
{
    Form1->Color = clGreen;
}
//---------------------------------------------------------
```

要注意的是,如果此前您在事件句柄 Button1Click 中写了代码,而且还用老名字引用了该 Form,则当您修改 Form 的名称时,这部分代码不会自动改变,因为这些代码是您自己键入的。这时候,您就要手动更改对 Form 的引用,否则编译不能通过。

## 1.3 对象的作用域问题

在 C++ Builder 3 中要特别注意对象的作用域问题,因为这关系到对象的成员是否可见和可访问。总的原则是,在类的方法中,类的成员都是可见的。例如,假设 Form(类名叫 TForm1)上有一个按钮 Button1,由于 C++ Builder 3 把处理 Form 及 Form 上元件事件的句柄作为 TForm1 的一个方法,因此,在处理按钮的 OnClick 事件的句柄中,您就可以任意访问 TForm1 的所有成员,而不需要用 TForm1 的变量来引用,例如:

```
__fastcall TForm1::Button1Click(TObject *Sender);
{
    Color = clFuchsia;
    Button1->Color = clLime;
}
```

由于 Color 是 TForm 的成员,所以也就成为了 TForm1 的成员,因此,您可以直接对 Color特性赋值,而不需要像下面这么写(尽管这样写编译器也不认为出错):

```
Form1->Color = clFuchsia;
```

但是,如果要访问 Form 上某个元件的成员,例如 Button1 的 Color 特性,您就必须加上对象限定符,否则编译器将认为您要访问的是 TForm1 的 Color 特性。

也许您要问,既然访问按钮的 Color 特性时要加上 Button1 限定符,那为什么不在

Button1前加 Form1 限定符呢？这是因为，Button1 被声明为 TForm1 的一个数据成员，而在类的方法中，类的成员总是可见的、可访问的，因此，不需要加 Form1 限定符。

假设当前项目中有两个 Form，分别是 Form1 和 Form2，如果要在 Form1 中访问 Form2 上的 Button2 元件，该怎样访问呢？

由于 Button2 已超出 Form1 的作用域，因此，在访问 Button2 时您必须加上 Form2 限定符，程序示例如下：

```
Form2 -> Button2 -> Color = clLime;
```

如果您以为光这样就可以了，那就错了，因为 Form1 还根本就不知道 Form2 是什么东西。因此，您还需要把 Form2 的头文件包含到 Form1 的单元文件中。

## 1.4 类成员的可见性

面向对象编程的重要特征之一就是隐藏复杂性。C++ Builder 3 通过__published、private、public、protected 和 automated 等几个关键字使类成员具有不同的可见性，例如：

```
//---------------------------------------------------------
class TForm1 : public TForm
{
    __published:
    TButton *Button1;
    void __fastcall Button1Click(TObject *Sender);
    private:
    public:
    __fastcall TForm1(TComponent* Owner);
};
```

上例中，数据 Button1 和方法 Button1Click 是在__published 部分声明的，而 TForm1 的构造函数是在 public 部分声明的，那么这几个关键字究竟是什么含义呢？

在 public 部分声明的成员是公共的，也就是说，它们虽然是在某个类中声明的，但其他类的实例也可以访问。例如，假设应用程序由两个 Form 构成，相应的单元是 Unit1 和 Unit2，您希望 Unit2 能共享 Unit1 中的整型变量 Count，那么您可以把 Count 在 TForm1 类中的 public 部分声明，然后把 Unit1 的头文件包含到 Unit2 中。

不过，除非您必须把某个成员在不同类之间共享，一般来说尽量不要把成员声明放在类的 public 部分，以防止程序意外地不正确地修改了数据。

在 private 部分声明的成员是私有的，它们只能被同一个类中的方法访问，就好像局部变量一样。对于其他类(包括它的派生类)，private 部分声明的成员是不可见的，这就是面向对象编程中的数据保护机制。程序员不必知道类实现的细节，只需关心类的接口。

protected 与 private 有些类似。在 protected 部分声明的成员是私有的，不同的是，在 protected 部分声明的成员在它的派生类中是可见的，并且将成为派生类的私有成员。

在 protected 部分声明的成员通常是方法，这样既可以在派生类中访问这些方法，又不必知道方法实现的细节。

在 __published 部分声明的成员，其可见性与在 public 部分声明的成员的可见性是一样的，它们都是公共的。不同的是，__published 主要用于声明 Form 上的元件和事件句柄，它是由 IDE 自动维护的。

在 automated 部分声明的成员，其可见性与在 public 部分声明的成员的可见性是一样的，它们都是公共的。唯一的区别在于，在 automated 部分声明的方法和特性将生成 OLE 自动化的类型信息。不过，在 C++ Builder 3 中，automated 的作用不大。

## 1.5 对象的相互赋值

尽管类是一种复杂的数据类型，既有数据又有方法，但是，您完全可以像对一般的变量那样，用赋值号把一个类的变量赋给另一个类的变量。只要这两个类的类型是一致的或相容的就行。例如，假设 Form1 和 Form2 都是 TForm 的变量，您可以这么写

```
Form2 = Form1;
```

因为 Form1 和 Form2 的类型都是 TForm。

您还可以把派生类的变量赋给基类的变量。例如，假设 TMyForm 是这样声明的：

```
class TMyForm : public TForm
{
    __published:
    TButton *Button1;
    TEdit *Edit1;
    TDBGrid *DBGrid1;
    TDatabase *Database1;
    private:
    public:
    virtual __fastcall TMyForm(TComponent * Owner);
};
```

然后分别声明了 TMyForm 和 TForm 的变量：

```
TMyForm *MyForm;
TForm *AForm;
```

则由于 TMyForm 是从 TForm 继承下来的，因此您可以这样赋值：

```
AForm = MyForm;
```

不知您注意到没有，每一个事件句柄中都有一个 Sender 参数，它的类型是 TObject。大家知道，VCL 的所有对象都是从 TObject 继承下来的，因此，VCL 的所有对象都能赋值给 Sender 参数。这就是为什么 Sender 参数能代表所有 VCL 对象的原因。

Sender 参数主要用于在运行期判断元件的类型，程序示例如下：

```
if (typeid( * Sender) == typeid(TEdit))
    DoSomething;
else
    DoSomethingElse;
```

## 1.6 自己创建一个对象

C++ Builder 3 已经在 VCL 中声明了众多的对象。不过,有时候您仍可能需要自己声明一个类,然后再创建类的实例,即对象,因为 VCL 中的对象未必能完全满足您的编程需要。例如,您可以声明一个 TEmployee 类:

```
class TEmployee : public TObject
{
    private:
    char Name[25];
    char Title[25];
    double HourlyRate;
    public:
    double CalculatePayAmount();
    TEmployee();
};
```

类的声明一般放在头文件中,然后您得在要用到这个类的单元文件中声明类的变量:

```
TEmployee *Employee;
```

TEmployee 只是一种数据类型,要使用这个类,您必须创建类的实例:

```
Employee = new TEmployee;
```

new 会自动调用 TEmployee 的构造函数。现在您就可以访问 TEmployee 中的成员了。

前面讲过,对象实际上是一块内存区域,换句话说就是 Windows 的资源。如果您不再需要用到 TEmployee 的对象,应当及时地删除它,以释放它所占用的资源。

要删除 TEmployee 的对象,您可以调用 delete:

```
delete Employee;
```

不过,您要考虑到这样一种情况,就是如果程序出现异常,程序将非正常退出,这种情况下,程序可能执行不到删除对象的地方。为了保证资源能可靠地得到释放,您就要用到 C++ 的异常处理机制,把删除对象的代码放到 try...catch 块中。

要说明的是,当您把元件选项板上的元件加到 Form 上时,C++ Builder 3 会自动创建元件的实例;当程序正常退出的时候,C++ Builder 3 会自动删除元件的实例。这一切都用不着您自己编写代码,除非您是自己在运行期动态地把一个元件加到了 Form 上。

## 1.7 VCL 的结构

VCL(Visual Component Library 的缩写)是 C++ Builder 3 的核心。VCL 是完全面向对象的,VCL 中的所有对象都存在着继承与被继承的关系,下页图是 VCL 的示意。

从示意图可以看出,最顶层的是 TObject,它是一切对象的基类。